Exploiting Graph Properties of Game Trees

Aske Plaat[*,1], Jonathan Schaeffer[2], Wim Pijls[1], Arie de Bruin[1]
plaat@theory.lcs.mit.edu, jonathan@cs.ualberta.ca, whlmp@cs.few.eur.nl, arie@cs.few.eur.nl

[1] Erasmus University, Dept. CS,
Room H4-13, P.O. Box 1738
3000 DR Rotterdam, The Netherlands

[2] University of Alberta, Dept. CS
615 General Services Building
Edmonton, AB, Canada T6G 2H1

Abstract

The state space of most adversary games is a directed graph. However, due to the success of simple recursive algorithms based on Alpha-Beta, theoreticians and practitioners have concentrated on the traversal of *trees*, giving the field the name "game-tree search." This paper shows that the focus on trees has obscured some important properties of the underlying graphs. One of the hallmarks of the field of game-tree search has been the notion of the minimal tree, the smallest tree that has to be searched by *any* algorithm to find the minimax value. In fact, for most games it is a directed graph. As demonstrated in chess and checkers, we show that the minimal graph is significantly smaller than previously thought, proving that there is more room for improvement of current algorithms. We exploit the graph properties of the search space to reduce the size of trees built in practice by at least 25%. For over a decade, fixed-depth Alpha-Beta searching has been considered a closed subject, with research moving on to more application-dependent techniques. This work opens up new avenues of research for further application-independent improvements.

Content areas: Search, Game playing.

Introduction

Search is a topic of fundamental importance to artificial intelligence (AI). The range of search strategies investigated stretch from application-independent methods to application-dependent, knowledge-intensive methods. The former has the promise of general applicability, the latter of high performance.

An important experimental domain for search algorithms has been the field of game playing. Arguably, this research has been one of the most successful in AI, leading to impressive results in chess (Deep Blue, formerly Deep Thought, playing at Grandmaster strength (Hsu *et al.* 1990)), checkers (Chinook, the World Man-Machine Champion (Schaeffer *et al.* 1996)), Othello (Logistello, significantly stronger than all humans (Buro 1994)), and Backgammon (TD-Gammon, playing at World Championship level strength (Tesauro 1995)).

Applications of two-player, adversary search was one of the initial goals of the fledgling field of AI and continues to be the source for many innovative ideas. The Alpha-Beta algorithm is at the heart of most high-performance game-playing programs. Knuth and Moore showed that there exists a minimal tree that has to be searched by any algorithm to prove the minimax game value (Knuth & Moore 1975). For a tree of fixed branching factor w and fixed depth d, there is a large gap between the worst case ($O(w^d)$) and the best case, the minimal tree ($O(w^{d/2})$), prompting extensive research on methods to come close to the best case. These are mostly application-independent search techniques. They are so successful that high performance programs are reported to build search trees within a factor of 1.5 of the minimal tree (Ebeling 1987). This is a surprising result, given that the basic search structure is based on application-independent methods that use no explicit knowledge of the application domain.

The data structure traversed during a search is usually mistakenly labeled as a tree in the literature, when in fact it is a directed (acyclic) graph. That the trees are really graphs can be seen by the use of transposition tables. These tables cache search results in the event that the same node is revisited, possibly via another path. When one starts thinking of the trees as directed graphs, other application-independent methods become possible, as the remainder of this paper will show.

We discuss a number of application-independent search methods for achieving significant improvements to Alpha-Beta searching. This paper has the following contributions:

- *Minimal Tree:* Search efficiency is usually compared against Alpha-Beta's best case. By exploiting graph properties of the search space, we show that the real minimal graph that defines the minimax value is significantly smaller than was previously assumed. This implies that there is more room for improvement of fixed-depth full-width adversary search algorithms.
- *Transpositions:* This paper introduces the ETC enhancement to maximize the utility of cached information seen earlier in the search. By directing the search towards previously visited nodes, the program can maximize information reuse, and thereby reduce the number of nodes visisted. For chess, ETC improves search efficiency by 28%.
- *Exploiting Irregular Branching Factor:* Given that nodes

[*] Current address: MIT LCS, 545 Tech Square, Cambridge, MA 02139, USA

in the search space do not necessarily have the same out degree (branching factor), it is possible to exploit this property by directing the search in a least-work-first fashion (as seen, for example, in (Allis, van der Meulen, & van den Herik 1994; McAllester 1988)). In a way, this is the small-is-quick approach from single-agent optimization (Pearl 1984). This paper demonstrates the potential for least-work-first search reductions. Initial results for checkers show that some gains (8%) are possible.

- *New Application-Independent Search Techniques:* Since the success of the work on minimal search windows (Finkel & Fishburn 1982; Fishburn 1981; Pearl 1984; Reinefeld, Schaeffer, & Marsland 1985) and move ordering (the history heuristic (Schaeffer 1983; 1989)) in the early 1980's, researchers have become convinced that fixed-depth search is close to its optimum. Consequently, most research has concentrated on application-dependent techniques such as knowledge-intensive move ordering and selective searches. Many of the application-independent search techniques that originated in adversary search have found their way to other domains (for example, iterative deepening and transposition tables in single-agent search (Kaindl *et al.* 1995; Korf 1985; 1990; Reinefeld & Marsland 1994)). The techniques that are introduced in this and another work (Plaat *et al.* 1995; 1996), are the first in more than a decade to show that new application-independent methods can still improve search efficiency.

Intuitively, one would expect that knowledge-intensive approaches would be effective for solving hard real-time problems, such as playing grandmaster-level chess. The success of application-independent search techniques (brute-force) is amazing. This paper strengthens the brute-force approach by introducing new methods.

The Minimal Tree

For fixed-depth adversary or minimax search trees, a so-called minimal tree exists that must be searched in order to find the minimax value (Knuth & Moore 1975). For a tree of uniform width w and depth d, Knuth and Moore proved that the number of leaves in the minimal tree is $w^{\lfloor d/2 \rfloor} + w^{\lceil d/2 \rceil} - 1$, approximately the square root of the size of the minimax tree. The minimal tree coincides with Alpha-Beta's best case. Alpha-Beta achieves this best case scenario only under the exceptional condition that at every node where a cutoff can occur, the cutoff is achieved by the first move considered (i.e., the children of a node are re-ordered so that a "best" move is always considered first).

The minimal tree is a natural performance standard for minimax search algorithms. Many researchers judge the quality of new algorithms and heuristics by comparing the size of the search tree built to it. Numerous simulations of minimax search algorithms have been performed using a comparison with the size of the minimal tree as the performance metric. Since each node in the simulated artificial trees has only one parent, and a fixed w and d are used, the minimal tree is trivially calculated by the Knuth and Moore formula. The simulations invariably conclude that, given a good move ordering, Alpha-Beta variants are performing almost perfectly, since the tree size observed is close to the size of the minimal search tree (see for example, (Campbell & Marsland 1983; Kaindl, Shams, & Horacek 1991; Marsland, Reinefeld, & Schaeffer 1987; Muszycka & Shinghal 1985)).

For real-world applications, calculating the size of the minimal tree is complicated by the presence of transpositions (nodes with more than one parent) and a variable width w. To approximate the minimal tree (really a graph), one can build the best-case Alpha-Beta scenario and use it as the performance standard. This can be done using a two-stage procedure, defined by Ebeling (Ebeling 1987):

1. *Initial search:* Perform an Alpha-Beta search, saving the best move at each node in a table.
2. *Re-search:* Re-search the tree using the table as an oracle that "knows" the best move to consider.

The size of this tree is a good approximation of the minimal tree. (For the sake of consistency we will in this section continue to call this graph the minimal tree.)

Numerous game-playing programs have been shown to perform quite close to the best-case Alpha-Beta minimal tree. For example, in chess, *Belle* is reported to be within a factor of 2.2 of the minimal Alpha-Beta tree (Ebeling 1987), *Phoenix* within 1.4 (measured in 1985) (Schaeffer 1986), *Hitech* within 1.5 (Ebeling 1987) and *Zugzwang* within 1.2 (Feldmann 1993). These results suggest that there is little room for improvement in fixed-depth Alpha-Beta searching.

As an independent check of the effectiveness of Alpha-Beta search, we conducted measurements using two tournament-quality game-playing programs, Chinook (checkers) (Schaeffer *et al.* 1992) and Phoenix (chess) (Schaeffer 1986). This covers the range of high (35 in chess) to low (3 in checkers) average branching factors. The two programs use the NegaScout variant of Alpha-Beta (Reinefeld 1983) enhanced with aspiration windows, the history heuristic, iterative deepening and transposition tables (Schaeffer 1989) (2^{21} entries—large enough to store the minimal tree). We will refer to this search algorithm with its enhancements as enhanced NegaScout (EnhNS). The programs were modified to search to a fixed depth, ensuring that changes to the search parameters would not alter the minimax value of the graph. Both programs have presumably been finely tuned and achieve high performance. Data points were averaged over a set of 20 test positions (some data points were averaged over 40 test positions for verification). The different branching factors of the two games affect the depth of the search trees built within a reasonable amount of time. For checkers, our experiments were to 17 ply deep (one ply equals one move by one player) and for chess, 8 ply.

The results of comparing enhanced NegaScout against the minimal tree are shown in figure 1 (based on all nodes searched in the last iteration). The figure confirms that the best programs are searching close to the minimal tree (within a small factor).

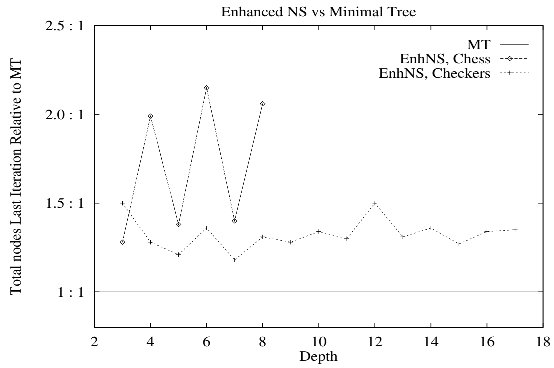

Figure 1: Efficiency of Programs Relative to the Minimal Tree

An interesting feature is that both programs, chess in particular, have significantly worse performance for even depths. The reason for this can be seen if we look at the structure of the minimal tree (the parity of the depth influences the number of nodes where a cutoff is expected). This leads to an important point: reporting the efficiency of a fixed-depth search algorithm based on odd-ply data is misleading. The odd-ply iterations give an inflated view of the search efficiency. For odd-ply searches, both programs are searching with an efficiency similar to the results reported for other programs. However, the even-ply data indicates that, at least for chess, there is still room for improvement.

To summarize, the evidence shows that search efficiency is close to the minimal tree. These facts have convinced many researchers that there is almost no room left for improving application-independent methods for fixed-depth Alpha-Beta. Consequently, the field of game-tree search has moved on to more application-dependent methods, reducing the generality of the research in game-tree search.

The Minimal Graph

The previous section discussed the Knuth and Moore minimal tree as a metric for comparing algorithms that search trees in a left-to-right, depth-first manner. However, this minimal tree is not necessarily the smallest possible. Alpha-Beta stops as soon as it has found a cutoff. It could very well be that an alternative move could also achieve a cutoff with less work.

In real applications, where w is not uniform, the minimal tree defined in the previous section is not really minimal, because at cutoff nodes no attempt has been made to achieve the cutoff with the smallest search effort. Further, the minimal tree is not really a tree; many applications allow nodes to have more than one parent. The minimal search graph must also try to maximize the re-use of previously visited nodes. Ebeling's procedure to compute a "minimal graph" really yields a *left-first* minimal graph (LFMG), since only the left-most move causing a cutoff is investigated. The *real* minimal graph (RMG) must select the cutoff move leading to the smallest search.

The preceding suggests modifying Ebeling's method to always get the cheapest cutoffs:

1. *Initial search:* Perform an Alpha-Beta search. At cutoff nodes, continue to examine alternative moves so that the "cheapest" cutoff (smallest search tree) is found. Save the best move at each node in a table.
2. *Re-search:* Re-search the tree using the table as an oracle that "knows" the best move to consider.

Step 1 can be optimized to stop the search of a cutoff candidate as soon as its subtree size exceeds the size of the current cheapest cutoff.

A complication is that in the presence of transpositions, the size of a search can be largely influenced by the frequency of transpositions. The size of a move's subtree can be influenced by whether another move has been searched or not, since the subtrees may have some nodes in common. This problem is known as the interacting sub-goal problem in AND/OR graphs (Nilsson 1980; Pearl 1984). Thus this procedure can only compute an upper bound on the size of the RMG and, hence, we call it the ARMG (approximate real minimal graph).

Without an oracle, computing the RMG is intractable. Even computing the ARMG is infeasible since it is asymptotic to full minimax search. Here we present a method for finding some of the cheaper cutoffs, allowing us to obtain an upper bound on the ARMG. This construction is done by performing the ARMG algorithm on the lowest d plies of the tree only. This variant is referred to as ARMG(d).

The approach still suffers from the interacting sub-goal problem. Therefore, an extra search must be performed to count the real size of the approximated minimal graph.

The results for Chinook and Phoenix are shown in figures 2 and 3 respectively. The results were obtained using ARMG(3). The reduction in the LFMG is most noticeable in checkers. Checkers has a variable branching factor (averaging 1.2 moves in a capture position and 7.8 in a non-capture position), while chess is relatively uniform at 35–40 moves per position (moving out of check is an exception, but those positions occur relatively rarely in the search). Thus there is more potential for cheaper cutoffs in checkers, as the graphs show.

The conclusion is that the Knuth and Moore minimal tree and the Ebeling approximation of the minimal graph are loose bounds on the real minimal graph; the RMG could be much smaller. For chess, the LFMG computed by Ebeling's procedure is 25% larger than our upper bound on the RMG; for checkers it is an amazing 122% (it is 2.22 times as large). In effect, high-performance game-playing programs are not searching quite as efficiently as has been assumed.

Exploiting Graph Properties for Real-time Search

The preceding section showed that the real minimal graph is smaller than the left-first minimal graph. These results were computed using off-line methods which, in some cases, took many days to compute. Can some of this experience be distilled into new methods for improving real-time, on-line

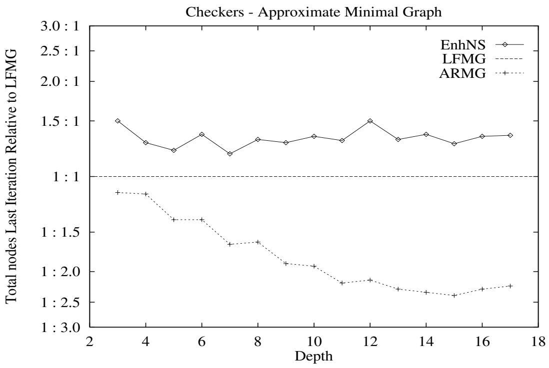

Figure 2: LFMG Is Not Minimal, Checkers

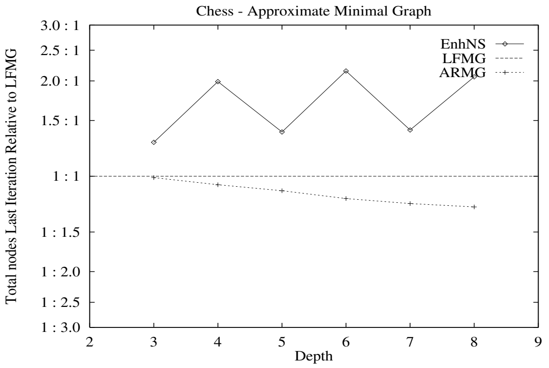

Figure 3: LFMG Is Not Minimal, Chess

Alpha-Beta searching? Computing the ARMG showed the importance of two items:

1. *Maximization of information reuse:* The search space is really a graph, meaning that a node can have more than one parent. We want to encourage the search to visit previously seen nodes to re-use available information.
2. *Least work first:* There may be alternative moves that can achieve a cut-off. If possible, assess potential cutoff moves based on the size of the search tree they are likely to build.

Both ideas translate into practical enhancements for real-time Alpha-Beta searching.

A simple and relatively cheap enhancement to improve search efficiency is to try and make more effective use of the transposition table. Consider interior node *N* with children *B* and *C* (figure 4). The transposition table suggests move *B* and, as long as it produces a cutoff, move *C* will never be explored. However, node *C* might transpose into a part of the tree, node *A*, that has already been analyzed. Before doing any search at an interior node, a quick check in the transposition table of all the positions arising from this node may result in finding a cutoff. *Enhanced Transposition Cutoffs*, ETC, performs transposition table lookups on successors of a node, looking for transpositions into previously searched

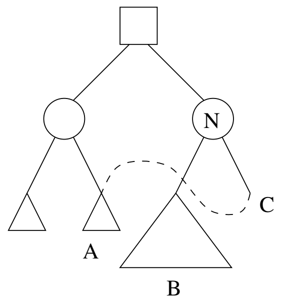

Figure 4: Enhanced Transposition Cutoff

lines. In effect, in a left-to-right search, ETC encourages subtrees in the right part of the tree to transpose into the left.

Figure 5 shows the pseudo-code of ETC embedded in Alpha-Beta. The ETC specific part is marked by **. In the figure, *n* is a node, f^+ is an upper bound on its value and f^- is a lower bound. *Retrieve*(*n*) extracts information on node *n* from the transposition table, while *store*(*n*) saves it. *Eval* evaluates a leaf node in the tree. At interior nodes, the moves are generated using *generatemoves*(*n*) and examined using *firstchild* and *nextchild*.

Figure 6 shows the results of enhancing Phoenix with ETC. For search depth 8, ETC lowered the number of expanded total nodes by 28% compared to enhanced NegaScout. Figure 7 shows that for Chinook ETC improves the search by 22%. For both programs, the reduction in search tree size offered by ETC is, in part, offset by the increased computation per node. Performing ETC at all interior nodes is too expensive. A compromise, performing ETC at all interior nodes that are more than 2 ply away from the leaves, results in most of the ETC benefits with only a small computational overhead. With this modification, the percent that ETC reduces the tree size translates into a slightly smaller percent reduction in execution time (graphs not shown). Thus, ETC is a practical enhancement to most Alpha-Beta search programs.

Checkers has a variable branching factor implying that there can be large differences in the search effort required to achieve a cutoff. The off-line version found cheaper cutoffs by doing a partial minimax search, continuing the search after a cutoff had been found. This is obviously too computationally intensive for real-time usage. We explored a number of different ways for predicting moves that lead to cheaper cutoffs. The method reported here, called Exploiting Irregular Branching factor, EIB for short, involves ordering moves based on the branching factor of the position that they lead to. In checkers, this means favoring moves that lead to a capture position. Moves that yield a position with a small branching factor are more favorably biased than those that lead to a larger branching factor. This bias is implemented as a bonus score added to the history heuristic for that move. After the move suggested by the transposition table (if available) is searched, the remaining moves are considered in the order of highest to lowest

```
   function alphabeta-ETC-EIB(n, α, β) → g;
     /* Check in transposition table */
     if retrieve(n) = ok then
       if n.f⁻ ≥ β then return n.f⁻;
       if n.f⁺ ≤ α then return n.f⁺;
     if n = leaf then g := eval(n);
     else
       m := generatemoves(n);
       /* Look for ETC */
**     c := firstmove(m);
**     while c ≠ ⊥ do
**       if retrieve(c) = ok then
**         if n is a max node and c.f⁻ ≥ β then return c.f⁻;
**         if n is a min node and c.f⁺ ≤ α then return c.f⁺;
**       c := nextmove(m);
       /* Order moves using EIB */
++     ordermoves(m);
       /* Standard Alpha-Beta */
       if n is a max node then
         g := −∞; a := α;
         c := firstmove(m);
         while g < β and c ≠ ⊥ do
           g := max(g, alphabeta-ETC-EIB(c, a, β));
           a := max(a, g);
           c := nextmove(m);
       else /* n is a min node */
         g := +∞; b := β;
         c := firstmove(m);
         while g > α and c ≠ ⊥ do
           g := min(g, alphabeta-ETC-EIB(c, α, b));
           b := min(b, g);
           c := nextmove(m);
     /* Save result in transposition table */
     if g < β then n.f⁺ := g;
     if g > α then n.f⁻ := g;
     store n.f⁻, n.f⁺;
     return g;
```

Figure 5: ETC-EIB Pseudo-Code

history heuristic score. This enhancement is hidden in the *ordermoves* routine in figure 5 (indicated by a ++).

Figure 7 shows the results of exploiting the irregular branching factor in checkers. Enhanced NegaScout benefits 8% with EIB. The combination of ETC and EIB yields a cumulative 28% reduction.

The cost of performing EIB is high, while the savings are small. Various schemes for cutting the cost of EIB (such as restricting the set of nodes where EIB is performed) only result in a small improvement in execution time.

ETC and EIB are application independent, but their effectiveness is a function of the properties of the game. For example, chess has many transpositions and, consequently, ETC is very effective. However, the branching factor is fairly uniform implying EIB is probably not useful. Checkers also has many transpositions, but also has an irregular branching factor. Thus both ETC and EIB are effective.

We have also experimented with the game of Othello. The LFMG is roughly 42% larger than the ARMG(2) showing that there is room for improvement. Transpositions in the game occur relatively infrequently (because moves can make dramatic changes to the board), limiting the benefits of ETC (to roughly 4%). The branching factor is generally uniform (around 10) but becomes irregular as the end of game approaches. We have not experimented with EIB in Othello.

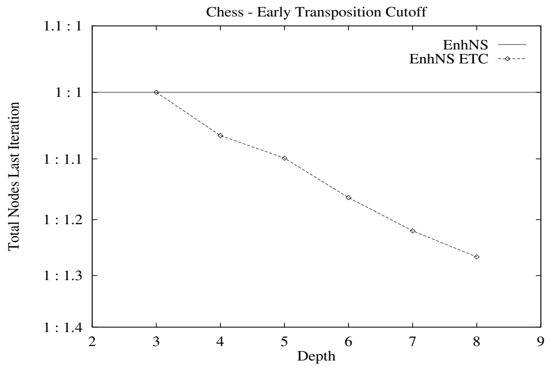

Figure 6: Effectiveness of Enhanced Transposition Cutoff in Chess

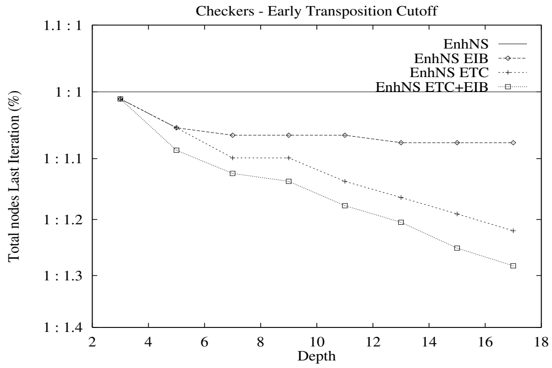

Figure 7: Effectiveness of Enhancements in Checkers

Conclusions

This research reports fixed-depth searching. When searching to variable depth (search extensions and forward pruning), the search *efficiency* savings of ETC and EIB cannot be quantified by node counting. However, tests with ETC (in Chinook and Deep Blue) show that the increased number of transpositions can improve the *accuracy* of the search. When ETC looks in the transposition table, it may find a position that has been searched deeper than the nominal search depth, allowing for a more accurate score to be used. On standard benchmark test suites, the ETC-enhanced program performs better (finds the correct move more often) than the non-ETC version.

Our results show that there is more room for finding new techniques in adversary search. This is in sharp contrast to the prevailing opinion which considered fixed-depth Alpha-

Beta searching to be a closed subject. The new methods that we introduce in this paper, together with MTD(f) (Plaat *et al.* 1996), are the first major improvements to fixed-depth Alpha-Beta searching since 1983. Our results warrant more research into application-independent methods, exploiting graph properties of the search space.

Acknowledgements

We thank Murray Campbell of the Deep Blue team for his feedback. This work has benefited from discussions with Mark Brockington, Yngvi Bjornsson, and Andreas Junghanns. The support of Jaap van den Herik, and the financial support of the Dutch Organization for Scientific Research (NWO), the Tinbergen Institute, the Natural Sciences and Engineering Research Council of Canada (grant OGP-5183) and the University of Alberta Central Research Fund are gratefully acknowledged.

References

Allis, L. V.; van der Meulen, M.; and van den Herik, H. J. 1994. Proof-number search. *Artif. Int.* 66:91–124.

Buro, M. 1994. *Techniken für die Bewertung von Spielsituation anhand von Beispielen*. Ph.D. Dissertation, Universität Paderborn, Germany.

Campbell, M. S., and Marsland, T. A. 1983. A comparison of minimax tree search algorithms. *Artif. Int.* 20:347–367.

Ebeling, C. 1987. *All the Right Moves*. Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA, MIT Press.

Feldmann, R. 1993. *Spielbaumsuche mit massiv parallelen Systemen*. Ph.D. Dissertation, Universität Paderborn, Germany.

Finkel, R. A., and Fishburn, J. P. 1982. Parallelism in alpha-beta search. *Artif. Int.* 19:89–106.

Fishburn, J. P. 1981. *Analysis of Speedup in Distributed Algorithms*. Ph.D. Dissertation, University of Wisconsin, Madison, WI.

Hsu, F.; Anantharaman, T. S.; Campbell, M. S.; and Nowatzyk, A. 1990. A grandmaster chess machine. *Scientific American* 263(4):44–50.

Kaindl, H.; Kainz, G.; Leeb, A.; and Smetana, H. 1995. How to use limited memory in heuristic search. In *Proc. of the 14th Int. Joint Conf. on Art. Int. (IJCAI-95)*, volume 1, 236–242.

Kaindl, H.; Shams, R.; and Horacek, H. 1991. Minimax search algorithms with and without aspiration windows. *IEEE Trans. on Pattern Anal. and Machine Intell.* 13(12):1225–1235.

Knuth, D. E., and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artif. Int.* 6(4):293–326.

Korf, R. E. 1985. Iterative deepening: An optimal admissible tree search. *Artif. Int.* 27:97–109.

Korf, R. E. 1990. Real-time heuristic search. *Artif. Int.* 42:189–211.

Marsland, T. A.; Reinefeld, A.; and Schaeffer, J. 1987. Low overhead alternatives to SSS*. *Artificial Intelligence* 31:185–199.

McAllester, D. A. 1988. Conspiracy numbers for min-max searching. *Artif. Int.* 35:287–310.

Muszycka, A., and Shinghal, R. 1985. An empirical comparison of pruning strategies in game trees. *IEEE Trans. on Systems, Man and Cybernetics* 15(3):389–399.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Company.

Pearl, J. 1984. *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Co.

Plaat, A.; Schaeffer, J.; Pijls, W.; and de Bruin, A. 1995. Best-first fixed-depth game-tree search in practice. In *Proc. of the 14th Int. Joint Conf. on Art. Int. (IJCAI-95)*, volume 1, 273–279.

Plaat, A.; Schaeffer, J.; Pijls, W.; and de Bruin, A. 1996. Best-first fixed-depth minimax algorithms. *Artif. Int.* To appear Fall 1996.

Reinefeld, A., and Marsland, T. A. 1994. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Anal. and Machine Intell.* 16(7):701–710.

Reinefeld, A.; Schaeffer, J.; and Marsland, T. A. 1985. Information acquisition in minimal window search. In *Proc. of the Int. Joint Conf. on Art. Int. (IJCAI-85)*, volume 2, 1040–1043.

Reinefeld, A. 1983. An improvement of the Scout tree-search algorithm. *ICCA Journal* 6(4):4–14.

Schaeffer, J.; Culberson, J.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1992. A world championship caliber checkers program. *Artif. Int.* 53(2-3):273–290.

Schaeffer, J.; Lake, R.; Lu, P.; and Bryant, M. 1996. Chinook: The world man-machine checkers champion. *AI Magazine*. To appear.

Schaeffer, J. 1983. The history heuristic. *ICCA Journal* 6(3):16–19.

Schaeffer, J. 1986. *Experiments in Search and Knowledge*. Ph.D. Dissertation, Department of Computing Science, University of Waterloo, Canada. Available as University of Alberta technical report TR86-12.

Schaeffer, J. 1989. The history heuristic and alpha-beta search enhancements in practice. *IEEE Trans. on Pattern Anal. and Machine Intell.* 11(1):1203–1212.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Comm. of the ACM* 38(3):58–68.